Advanced Structural and Chemical Imaging manuscript No.
(will be inserted by the editor)

An open-source software ecosystem for the interactive exploration of ultrafast electron scattering data

Laurent P. René de Cotret · Martin R. Otto · Mark J. Stern · Bradley J. Siwick

Received: date / Accepted: date

Abstract This paper details a software ecosystem comprising three free and open-source Python packages for processing raw ultrafast electron scattering (UES) data and interactively exploring the processed data. The first package, *iris*, is graphical user-interface program and library for interactive exploration of UES data. Under the hood, *iris* makes use of *npstreams*, an extensions of *numpy* to streaming array-processing, for high-throughput parallel data-reduction. Finally, we present *scikit-ued*, a library of reusable routines and data structures for analysis of UES data, including specialized image processing algorithms, simulation routines, and crystal structure manipulation operations. In this paper, some of the features or all three packages are highlighted, such as parallel data reduction, image registration, interactive exploration. The packages are fully tested and documented and are released under permissive licenses.

Keywords Ultrafast electron scattering · visualization · data processing · open-source

1 Introduction

The advent of ultrafast electron scattering (UES) has extended crystallography into the temporal dimension. Combining the spatial resolution of electron microscopy and femtosecond time-resolution, this laboratory-scale technique has shed light on a broad spectrum of phenomena, from photoinduced structural phase transitions in inorganic [1,2] and organic materials [3], to coherent nuclear motion in dilute molecular gas [4]. Beyond probing structure through elastic interactions, UES has provided a direct observation of electron-phonon couplings and phonon relaxation pathways in layered materials [5,6].

Ultrafast electron scattering is a stroboscopic technique. Electron scattering patterns are acquired a fixed time-delay after photoexcitation with an ultrafast laser. A full view of the scattering dynamics can be assembled by acquiring electron scattering patterns for sufficient time-delays. To maximize the signal-to-noise ratio, equivalent experiments – hereafter referred to as *scans* – are acquired sequentially. This set of equivalent sub-experiments forms the raw experimental data. In order to extract time-dynamics, the data must be distilled, or *reduced*; the data reduction might involve scattering intensity normalization or alignment, and ends in averaging. A physical picture of dynamics in a sample is built by looking at in scattering intensity over time, in a fixed region of reciprocal space. A conceptual view of the flow from raw data to dynamics is presented in figure 1.

The path from data acquisition to scientific insights can never be fully standardized and streamlined. However, one step common to all workflows is data exploration. This is especially important for time-resolved techniques such as UES due to the breadth and depth of information contained in a dataset. This emerging field would benefit greatly from an aggregation of efforts towards free and open-source tools that standardize and simplify analysis and interpretation of complex dynamics. In this work, we present a program built specifically for interactive data exploration, *iris*. We also introduce a software package of reusable routines and data structures related to ultrafast electron scattering (*scikit-ued*)

L. P. René de Cotret · Martin R. Otto · Mark J. Stern
Department of Physics, McGill University

Bradley J. Siwick
Departments of Physics and Chemistry, McGill University
E-mail: bradley.siwick@mcgill.ca

and a streaming array operations package (*npstreams*), establishing a software foundation on which the community can build. *Iris* is the first integrated solution for interacting with UES data.

The software ecosystem presented in this work is written in Python; the language is accessible to non-programmers, with a simple syntax and dynamic nature. The scientific Python community's dedication to good documentation and focus on free and open-source packages are attractive features that played heavily in the decision to write iris in Python. Performance bottlenecks can easily be rewritten in a more efficient, compiled language and seamlessly integrated with existing Python code [7]. The Python scientific stack is built on the *de facto* standard *numpy* package [8] that exports array operations rooted in compiled code, bypassing the slow nature of Python in most situations. Therefore, Python provides an ideal mix of performance and ease of development.

2 Interactive data exploration

The primary tool presented in this work is *iris*, first and foremost a graphical user-interface (GUI) program that allows for interactive exploration of ultrafast electron scattering data. From its GUI, users can interact with raw data, process this raw data into a reduced dataset, enabling interactive data exploration. *Iris*' second role consists of a library of data structures for handling ultrafast scattering data in external Python scripts and programs. Everything that can be done in the GUI, can also be done programmatically.

The first step in the typical *iris* workflow involves interacting with raw scattering patterns from a variety of possible unique formats. For this purpose, *Iris* supports plug-ins that act as bridges between arbitrary data formats for raw data and *iris*'s internal representation.

The second step in the typical *iris* workflow consists of data reduction. During this phase, equivalent scattering patterns acquired with the same time-delay are reduced in parallel. This reduction operation can involve image-alignment, intensity normalization, or any operation specified by a plug-in. scattering patterns are then concatenated into a three-dimensional array, similar to a video. Experimental metadata, such as sample temperature, photoexcitation conditions, electron energy, and more, are also preserved. Arbirary metadata can be defined by users through the use of plug-ins. The performance optimization of the data reduction pipeline is discussed in depth in section 3.

Iris handles large reduced datasets by storing them using the Hierarchical Data Format version 5 (HDF5), an archival format designed for large n-dimensional numerical datasets [9,10]. HDF5 supports transparent compression and data-corruption detection mechanisms. Most importantly, HDF5 supports data slicing, which allows to read specific portions of an HDF5 file without having to load the entire file – which can require tens of gigabytes of memory in the case of fully-reduced UES datasets.

The final stage of the exploration workflow – time-series extraction – is specific to time-resolved techniques. Thanks to HDF5's data slicing feature and careful performance optimizations within *iris*, dynamics in scattering patterns can be explored interactively, in real-time. Further data transformations are also possible, most importantly azimuthal averaging of scattering patterns from polycrystalline samples. Time-series extraction in the GUI is shown on figure 2 for two types of samples, single-crystal and polycrystal.

Iris was designed with compatibility in mind. For input compatibility, *iris*'s plug-in architecture allows to interface with arbitrary data formats. Writing a plug-in requires writing some Python code: users can then use the full power of the scientific Python stack to introduce extra processing in *iris*'s data reduction pipeline. In terms of output compatibility, *iris* datasets can be inspected and manipulated by a large variety of programs, thanks to HDF5's official and unofficial bindings to programming languages such as C/C++, Fortran77/Fortran90, LabVIEW, MATLAB, Mathematica, R, Julia, Python, and many more. The HDF5 layout is described in the online documentation.

3 Streaming operations on arrays

Raw scattering datasets can reach sizes up to hundreds of gigabytes. The reduction operation of concatenating images into dense *numpy* arrays, and then reducing - which usually involves scattering pattern alignment, intensity normalization, and finally a weighted average - is a slow process.

To effectively reduce the raw data, the package *npstreams* was created. *Npstreams* extends the core components of the array-oriented *numpy* package (called universal functions) to work on streams of arrays rather than dense, in-memory arrays. *Npstreams* can *automagically* generate streaming functions from *numpy* universal functions. The streams of arrays can be generated on-demand (such as images being progressively loaded from disk). Stream operations require much less memory, which in turns allows for parallelization. In fact, in the special case of stream operations on arrays of the same size (e.g. on images), data reduction can operate in constant-memory.

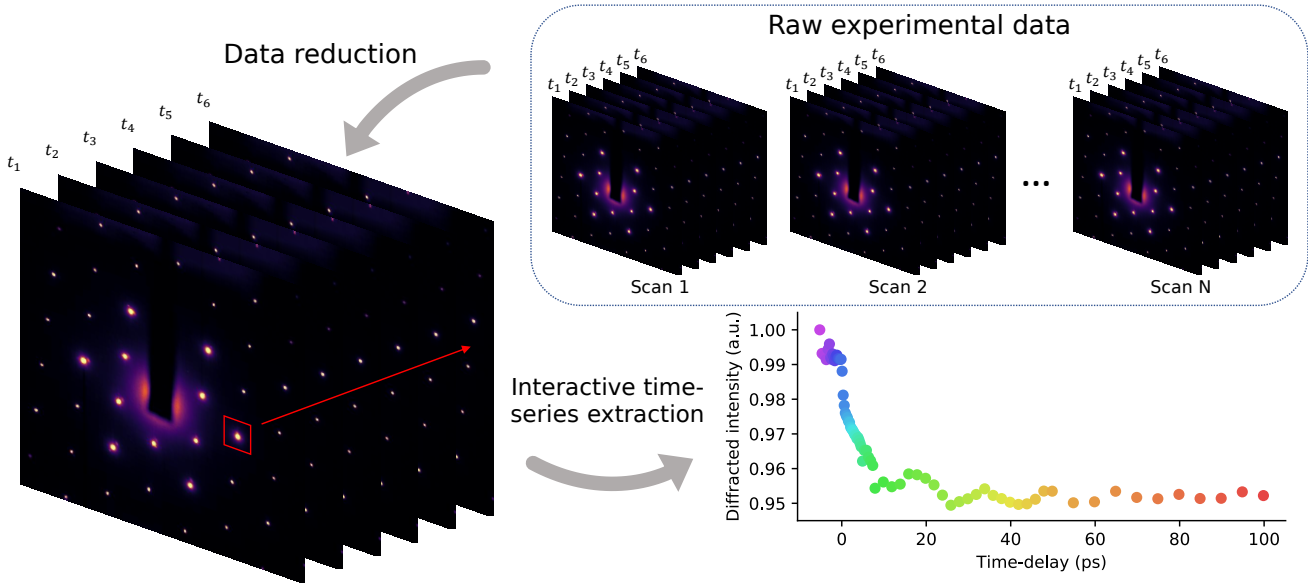

Fig. 1 Ultrafast electron scattering data exploration workflow. To maximize signals, ultrafast electron scattering experiments consists in many identical sub-experiments (or scans), as shown on the top right. Data reduction consists in combining these sub-experiments, which usually involves image-alignment, intensity normalization, and other corrections. Each individual time-delay can be reduced in parallel. The resulting reduced data is a stack of scattering patterns, similar to a video. Scattering intensity time-series can be extracted from either a single pixel or integrated over a groups of pixels (lower right)

In the case of data reduction in *iris*, a stream of arrays is formed by progressively loading scattering patterns at same time-delay (but from different scans). Therefore, the reduction of a dataset of many time-delays and many scans to a dataset of many time-delays and a single scan can be done in parallel. In the limiting case where data processing performance is not bound by data loading time, the performance increase due to *npstreams* alone is *linear* in the number of processing units. Thus, an 8-core computer would experience an 8x performance increase by using *npstreams* in parallel.

Single-core performance is also improved. To measure single-core performance, benchmarks set up a sequence of $n \times n$ arrays of random floating-point numbers representing scattering patterns. Then, arrays are either directly averaged using *npstreams*, or concatenated into a dense `ndarray` which is in turns averaged using *numpy*. The results of running this benchmark on sequence of varying lengths and arrays of varying sizes is presented in figure 3.

While the core of *npstreams* is concerned with autogenerating streaming functions from basic universal *numpy* functions, some more complex streaming functions are implemented, with the aim of providing real-world examples. For instance, a streaming weighted average and streaming weighted standard deviation routines are included, based on West [11].

```
>>> import npstreams as ns
>>> ns.benchmark()
********************************************************************************
                        NPSTREAMS PERFORMANCE BENCHMARK

    npstreams   1.5.3
    NumPy       1.14.3
    Speedup is NumPy time divided by npstreams time (Higher is better)
********************************************************************************
                       numpy.average vs npstreams.average

     shape = (4, 4)     speedup = 1.2707x
     shape = (8, 8)     speedup = 1.3454x
     shape = (16, 16)   speedup = 1.5626x
     shape = (64, 64)   speedup = 2.2815x
--------------------------------------------------------------------------------
                          numpy.sum vs npstreams.sum

     shape = (4, 4)     speedup = 1.2645x
     shape = (8, 8)     speedup = 1.3963x
     shape = (16, 16)   speedup = 1.5897x
     shape = (64, 64)   speedup = 2.3401x
--------------------------------------------------------------------------------
              numpy.add vs npstreams.reduce_ufunc(numpy.add, ...)

     shape = (4, 4)     speedup = 1.3326x
     shape = (8, 8)     speedup = 1.4502x
     shape = (16, 16)   speedup = 1.5985x
     shape = (64, 64)   speedup = 3.6624x
--------------------------------------------------------------------------------
                             ... (continued) ...
```

Listing 1: How to run *npstreams* benchmarks suite from the interactive Python interpreter. By default, a pre-selected set of functions from both the *numpy* and *npstreams* packages are compared.

Npstreams also includes benchmarking functionality, where users can generate benchmark results on their machine with a simple command, as shown in listing 1.

The functionality of *npstreams* extends beyond the requirements of *iris* and *scikit-ued*. Streaming array operations can deal with an infinite stream of arrays. For example, an operation can be applied until certain cri-

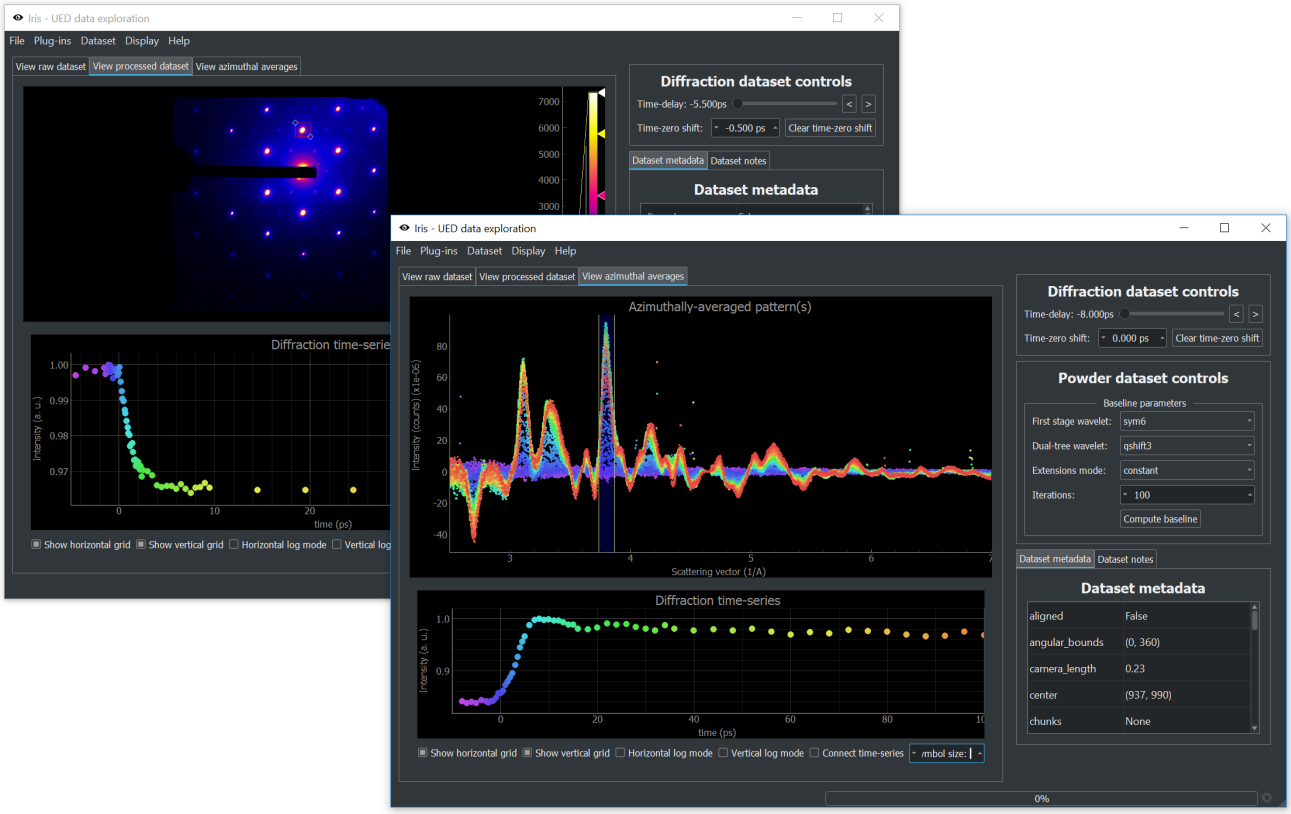

Fig. 2 Overview of the GUI component of *iris*. Two GUI instances show typical datasets. On the top left, Bragg peak dynamics for photoexcited single-crystal data is shown. Diffracted intensity is integrated in the red square and its time-dependence is shown in the bottom panel. On the bottom right, azimuthally-averaged baseline-corrected polycrystalline scattering data is presented. The pre-photoexcitation scattering patterns have been subtracted so that dynamics are more evident. Diffraction patterns are color-coded based on their time-delay, shown below. Diffracted intensity is integrated inside the blue zone and its time dependence is again shown on the bottom panel. Both integration regions can be interactively dragged, updating the time-series in real-time.

teria are met, or a pipeline can be assembled which can run forever, in constant (low) memory.

4 Reusable routines and data structures for ultrafast electron scattering data analysis

The scientific Python community has access to over 50 research-oriented packages called *scikits*, extensions of the general-purpose SciPy package [12]. These software packages provide routines and algorithms for handling image processing (*scikit-image* [13]), performing machine-learning tasks (*scikit-learn* [14]), interacting with spectroscopy data (*scikit-spectra*), bio-informatics (*scikit-bio*), and much more. In the tradition of these scikits, we introduce *scikit-ued*, a collection of algorithms, routines, and data structures commonly needed for ultrafast electron scattering data interactions. Most operations *iris* performs on datasets are offloaded to *scikit-ued*.

Scikit-ued provides a repository of reusable components in many domains, such as image analysis, crystal structure manipulation, baseline-removal, simulation, common utilities, and more. Some of its features are described in the following sections.

4.1 Baseline-removal

The baseline-removal functionality provided by *scikit-ued* is the evolution of the approach previously published by the authors. Readers interested in the details are referred to [15]. Some extensions have been made, such as the ability to compute a background for 2D images using the iterative algorithm based on the discrete wavelet transform presented in [16]. This approach can be used to remove blemishes on images: by treating hot spots or defects as foreground, the baseline of the original image will show no hotspots. An example of this is shown in figure 4.

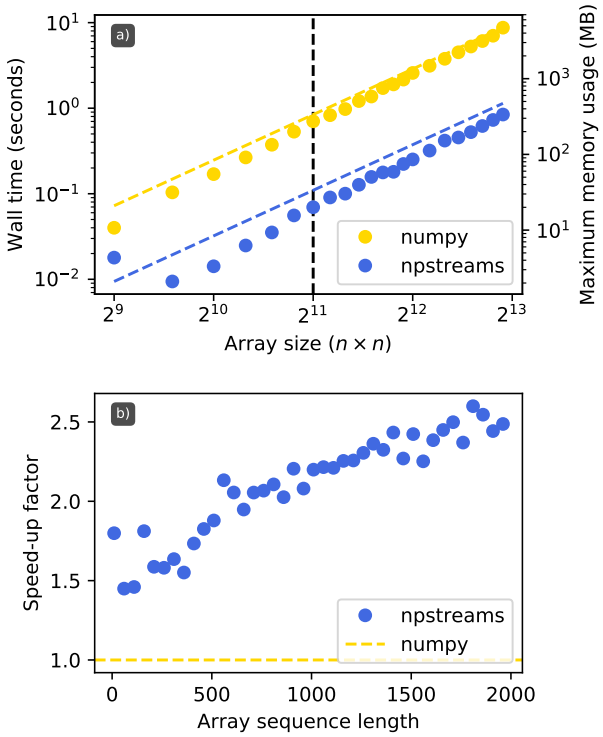

Fig. 3 Performance characterization of the *npstreams* package in comparison with *numpy*. The task was to average sequences of 2D arrays (representing scattering patterns). **a)** Wall time of averaging for a sequence of 10 arrays (solid) superimposed with maximum memory usage (dashed). The vertical dashed line marks the array size of 2048×2048, equivalent to a scattering pattern of 4 megapixels. **b)** Speed-up of using *npstreams* vs. *numpy* for averaging a sequence of arrays of size 512×512 elements.

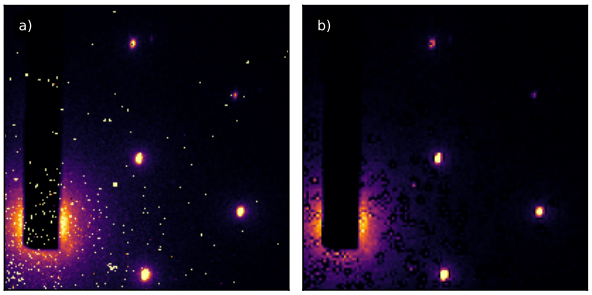

Fig. 4 Removing image hotspots using an iterative baseline-removal algorithm based on the discrete wavelet transform. **a)** Original scattering pattern shows hotspots due to laser light hitting the detector. **b)** Baseline of a). The hotspots are treated as the foreground by the iterative algorithm.

4.2 Image analysis

Scattering patterns analysis intersects with general image processing in many ways. In most cases, other libraries such as *scikit-image* can be used. Specific tasks not widely available elsewhere have been included in *scikit-ued*.

A common data processing task, implemented in the data reduction pipeline of *iris*, is scattering pattern alignment to a reference. During the course of data acquisition, scattering patterns can drift across the detector due to fluctuations in the electron beam alignment, while other image components (e.g. beam-stop, hard edges, detector defects) will appear static. Therefore, cross-correlation techniques used in image registration generally fail at scattering pattern alignment. *Scikit-ued* includes an advanced image alignment algorithm, known as the masked normalized cross-correlation image registration [17]. By masking static components of scattering data, the image registration algorithm will only compare misaligned sections, greatly enhancing registration accuracy. An example of this algorithm is presented in figure 5.

Symmetry-based operations are also implemented in *scikit-ued*. One such operation is azimuthal averaging, a procedure during which polycrystalline scattering patterns are reduced to one radial dimension. *Scikit-ued* includes discrete symmetry-based operations as well, most importantly discrete rotational averaging. scattering patterns exhibiting n-fold rotational symmetry can be transformed to yield a higher signal-to-noise ratio. This approach has recently been used to extract small ultrafast diffuse scattering signals from graphite [6]. An example of such symmetrization is presented in figure 6.

4.3 Structure manipulation

Scikit-ued includes data structures that make it easy to read and manipulate crystallographic information. The `Crystal` class is the primary data structure giving access to atomic positions, chemical composition, lattice information, and more. It can be generated from a few types of sources, with the most convenient type involving the ubiquitous Crystal Information File (CIF) format [18,19]. Structures can also be pulled from entries in the Crystallography Open Database [20,21], as well as generated from Protein Data Bank entries [22,23]. A number of structures (mostly simple elemental crystals) are included with the package. Finally, `Crystal` instances can be generated from (and converted to) the Atomic Simulation Environment `Atoms` format [24]. An

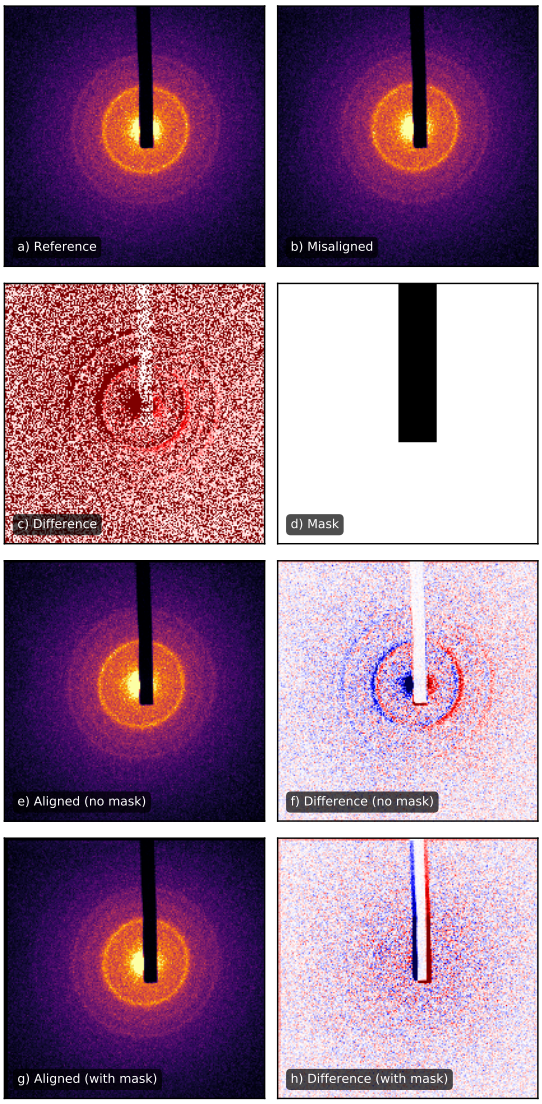

Fig. 5 scattering pattern alignment based on the Masked Normalized Cross-Correlation algorithm. **a)** Reference scattering patterns of polycrystalline chromium. **b)** Misaligned scattering pattern. **c)** Difference between the patterns in a) and b) shows structure indicative of a sideways shift. Note that the beam block has not moved. **d)** Pixel mask of the beam block. Black pixels represent zones to be ignored by the alignment procedure. **e)** Aligned image, registered without the mask in d). **f)** Difference between a) and e) shows residual misalignment. **g)** Aligned image, registered with the mask in d). **f)** Difference between a) and g) shows successful alignment.

example of structure manipulation is presented in listing 2. Once a `Crystal` instance has been created, space-group information and symmetry operations can be determined through the library spglib [26,27]. Using this information, crystals can also be reduced to their primitive cells.

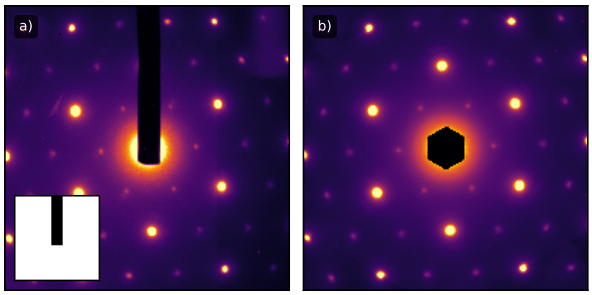

Fig. 6 Example of symmetrization of single-crystal scattering patterns. **a)**. Raw single-crystal scattering pattern. Inset shows the pixel mask indicating where the beam-block is located. Pixels under the mask are ignored during the symmetrization routine. **c)** 6-fold rotational symmetrization of scattering pattern in a) with beam-block masked reduces noise and enhances dynamics.

```
>>> from skued import Crystal
>>> c = Crystal.from_cod(9009089)  # Latest revision by default
>>> print(c)
< Crystal object with following unit cell:
    Atom O  @ (0.90, 0.79, 0.80)
    Atom O  @ (0.10, 0.21, 0.20)
    Atom O  @ (0.10, 0.29, 0.70)
    Atom O  @ (0.90, 0.71, 0.30)
    Atom O  @ (0.39, 0.69, 0.29)
    Atom O  @ (0.39, 0.81, 0.79)
    Atom O  @ (0.61, 0.19, 0.21)
    Atom O  @ (0.61, 0.31, 0.71)
    Atom V  @ (0.24, 0.53, 0.53)
    Atom V  @ (0.76, 0.48, 0.47)
    Atom V  @ (0.24, 0.97, 0.03)
    Atom V  @ (0.76, 0.03, 0.97)
Lattice parameters:
    5.743Å 4.517Å, 5.375Å, 90.00°, 122.60°, 90.00°
Source:
    COD num:9009089 rev:latest >
>>> print(c.spacegroup_info())
{'hall_number'         : 81,
 'hall_symbol'         : '-P 2ybc',
 'international_full'  : 'P 1 2_1/c 1',
 'international_number': 14,
 'international_symbol': 'P2_1/c',
 'pointgroup'          : 'C2h'}
>>> print(c.chemical_composition)
{'O': 0.666,
 'V': 0.333}
```

Listing 2: Generating a `Crystal` instance for VO_2 from the Crystallography Open Database entry 9009089 [25], inside the interactive Python interpreter.

4.4 Simulation

Scikit-ued has built-in support for calculation of crystal potentials, based on the parametrization of Kirkland [28]. The example of real-space, projected electrostatic potential of orthorhombic barium titanate ($BaTiO_3$) is shown in figure 7. These calculations are required in the implementation of multislice simulations [29,30].

From the parametrization of atomic potentials, the atomic form factors can also be calculated – and from them static structure factors. *Scikit-ued* exports a routine for simulating electron diffraction patterns for polycrystalline samples based on the atomic positions of `Crystal` objects. Examples of simulated diffraction pat-

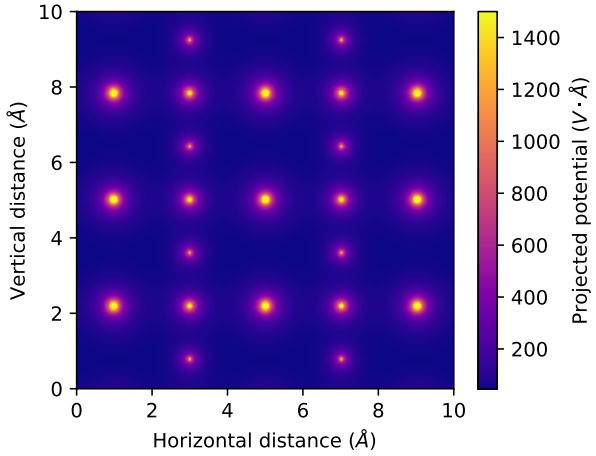

Fig. 7 Simulated electrostatic potential of orthorhombic barium titanate ($BaTiO_3$) projected onto the $z = 0$ plane. Structure file was taken from [31]

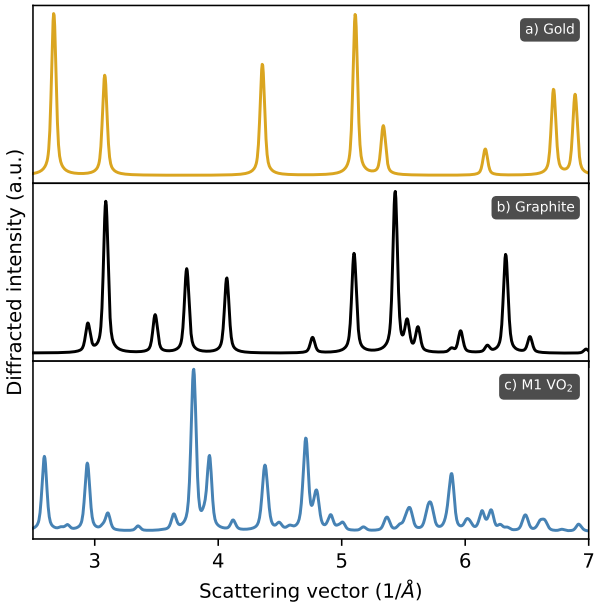

Fig. 8 Simulated polycrystalline electron diffraction patterns for gold, graphite, and monoclinic M1 VO_2.

terns from built-in *scikit-ued* structures are presented in figure 8.

4.5 Input and Output

Scikit-ued provides routines to read exotic image file formats. *Scikit-ued* can read Merlin Image Binary files (**.mib*) – both single image and multi-images files – as well as Gatan's proprietary DM3 and DM4 formats (**.dm3*, **.dm4*). *Scikit-ued* can also read all images supported by *scikit-image*, notably images encoded in the Tagged Image File Format (**.tiff*).

4.6 Miscellaneous

Many small utilities are included in *scikit-ued*. Fast computation of pseudo-voigt profiles is included and integrated with the polycrystalline diffraction simulation routine, based on the work by Ida et al. [32]. Calculation of thin film optical properties is also implemented, based on Tomlin [33]. Finally, a preliminary version of the Non-uniform Fast Fourier Transform (NFFT) is available [34]. Additional functionality is presented in the documentation.

5 Common features

All three libraries presented herein benefit from some common features and development tools.

First and foremost, all three packages are documented online (offline documentation is also available). Reference documentation is automatically generated from the source code, which limits the possibility of documentation being out-of-date with respect to the source code [35]. The documentation for all three packages also include hand-written tutorials. The documentation for each package is hosted online by Read the Docs and links are specified in section 8.1.

The repositories are hosted on GitHub, a web-based code hosting service with built-in version control through Git. As the packages are free and open-source, GitHub can be used to browse source code. It also provides features not available through Git itself, most importantly an issue tracker. This issue tracker is also directly accessible from the *iris* GUI through a help menu. Bugs and issues raised through GitHub are publicly visible and provide a place to discuss potential solutions.

After changes are committed to one of the repositories, the updated package is automatically installed and tested in a remote environment – a practice known as continuous integration. Committed changes also trigger the automatic generation of a new documentation version, which is then posted online within minutes.

Finally, stable versions of all three packages are uploaded to the Python Packaging Index (PyPI), where they can be inspected and downloaded. For users of the Anaconda Python distribution, the three packages are also available for install within the conda environment, which provides pre-compiled packages.

6 Roadmap

The natural progression for *iris* involves data exploration along different dimensions beyond time: diffracted intensity along dimensions of fluence, temperature, doping concentration, and more.
The target functionality of *npstreams* has largely been implemented. The obvious key improvement concerns performance, which could be increased by rewriting the core functionality in C, while exploiting *numpy*'s C interface.
Future developments of *scikit-ued* concern the simulation subpackage. Simulation of polycrystalline scattering pattern is indispensable as a tool to test methods and validate hypotheses; simulations of single-crystal scattering patterns is a natural evolution of *scikit-ued*'s capabilities. While the basic parametrizations of atomic properties is already implemented, as well as the computation of real-space electrostatic potential of arbitrary crystal structures (see figure 7), wave-propagation calculations – e.g. the multislice algorithm – remains to be implemented. See [36] for a summary of available electron scattering and microscopy simulation tools to which *scikit-ued* could bind.

7 Conclusion

An ecosystem of three free and open-source Python packages for exploring ultrafast electron scattering data was presented. This ecosystem, governed by permissive licenses, was created with collaboration in mind, while adhering to sane software development practice. We introduced *npstreams*, a streaming array processing library that allows for constant-memory data reduction. *Scikit-ued*, a *scipy* extension that contains algorithms and data structures for the analysis of UES data, was also presented. Finally, the ecosystem culminates in the *iris* package, the first integrated data-exploration interface specifically tailored to the data-rich nature of UES. *Iris*' plug-in functionality as well as real-time, interactive capabilities pushes the limits of what is possible in field of ultrafast electron scattering.

8 Declarations

8.1 Availability of data and materials

The data presented in this manuscript is publicly available in the *scikit-ued* repository.
The software packages' repositories are linked on the authors' homepage (`www.physics.mcgill.ca/siwicklab/software.html`). All packages are multi-platform and require the CPython interpreter version 3.6 or later. The *iris* and *scikit-ued* are licensed under the Massachusetts Institute of Technology (MIT) license, while *npstreams* is licensed under the Berkeley Software Distribution (BSD) 3-clause license (same as *numpy*).
The documentation for each package is hosted online by Read the Docs at the addresses located below:

Iris: `https://iris-ued.readthedocs.io/`
Scikit-ued: `https://scikit-ued.readthedocs.io/`
Npstreams: `https://npstreams.readthedocs.io/`

8.2 Competing interests

The authors declare that they have no competing interests.

8.3 Funding

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Fonds de Recherche du Québec - Nature et Technologies (FRQNT), and Canada Research Chairs (CRC) program.

8.4 Authors' contributions

LPRDC designed and implemented the software packages. LPRDC prepared the manuscript. MRO and MJS contributed data to showcase features of the software ecosystem. All authors read and approved the final manuscript.

Acknowledgements The authors would like to thank Samuel Palato and Hélène Seiler for their guidance concerning software design principles. The authors are also grateful to the contributors and maintainers of the several free and open-source packages on top of which this software ecosystem is built.

References

1. V.R. Morrison, R.P. Chatelain, K.L. Tiwari, A. Hendaoui, A. Bruhacs, M. Chaker, B.J. Siwick, Science **341**(6208), 19 (2014). DOI 10.1126/science.1253779
2. L. Waldecker, T.a. Miller, M. Rudé, R. Bertoni, J. Osmond, V. Pruneri, R.E. Simpson, R. Ernstorfer, S. Wall, Nature Materials **14**(July), 1 (2015). DOI 10.1038/nmat4359. URL `http://www.nature.com/doifinder/10.1038/nmat4359`
3. M. Gao, C. Lu, H. Jean-Ruel, L.C. Liu, A. Marx, K. Onda, S.Y. Koshihara, Y. Nakano, X. Shao, T. Hiramatsu, G. Saito, H. Yamochi, R.R. Cooney, G. Moriena, G. Sciaini, R.J.D. Miller, Nature **496**(7445), 343 (2013). DOI 10.1038/nature12044. URL `http://www.ncbi.nlm.nih.gov/pubmed/23598343`

4. J. Yang, M. Guehr, X. Shen, R. Li, T. Vecchione, R. Coffee, J. Corbett, A. Fry, N. Hartmann, C. Hast, K. Hegazy, K. Jobe, I. Makasyuk, J. Robinson, M.S. Robinson, S. Vetter, S. Weathersby, C. Yoneda, X. Wang, M. Centurion, Phys. Rev. Lett. **117**, 153002 (2016). DOI 10.1103/PhysRevLett.117.153002. URL https://link.aps.org/doi/10.1103/PhysRevLett.117.153002
5. L. Waldecker, R. Bertoni, H. Hübener, T. Brumme, T. Vasileiadis, D. Zahn, A. Rubio, R. Ernstorfer, Phys. Rev. Lett. **119**, 036803 (2017). DOI 10.1103/PhysRevLett.119.036803. URL https://link.aps.org/doi/10.1103/PhysRevLett.119.036803
6. M.J. Stern, L.P. René de Cotret, M.R. Otto, R.P. Chatelain, J.P. Boisvert, M. Sutton, B.J. Siwick, Phys. Rev. B **97**, 165416 (2018). DOI 10.1103/PhysRevB.97.165416. URL https://link.aps.org/doi/10.1103/PhysRevB.97.165416
7. L. Dalcin, R. Bradshaw, K. Smith, C. Citro, S. Behnel, D.S. Seljebotn, Computing in Science and Engineering **13**, 31 (2010). DOI doi.ieeecomputersociety.org/10.1109/MCSE.2010.118
8. S. van der Walt, S.C. Colbert, G. Varoquaux, Computing in Science Engineering **13**(2), 22 (2011). DOI 10.1109/MCSE.2011.37
9. T.H. Group. Hierarchical data format, version 5 (1997-2018). Https://www.hdfgroup.org/HDF5/
10. A. Collette, *Python and HDF5* (O'Reilly, 2013)
11. D.H.D. West, Commun. ACM **22**(9), 532 (1979). DOI 10.1145/359146.359153. URL http://doi.acm.org/10.1145/359146.359153
12. E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python (2001–). URL http://www.scipy.org/. [Online; accessed ¡today¿]
13. S. van der Walt, J.L. Schnberger, J. Nunez-Iglesias, F. Boulogne, J.D. Warner, N. Yager, E. Gouillart, T.a. Yu, PeerJ **2**, e453 (2014). DOI 10.7717/peerj.453. URL https://doi.org/10.7717/peerj.453
14. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Journal of Machine Learning Research **12**, 2825 (2011)
15. L.P.R. de Cotret, B.J. Siwick, Structural Dynamics **4**(4), 044004 (2017). DOI doi:10.1063/1.4972518. URL https://doi.org/10.1063/1.4972518
16. C.M. Galloway, E.C. Le Ru, P.G. Etchegoin, Applied Spectroscopy **63**(12), 1370 (2009). DOI 10.1366/000370209790108905
17. D. Padfield, IEEE Transactions on Image Processing **21**(5), 2706 (2012). DOI 10.1109/TIP.2011.2181402
18. T. Bjrkman, Computer Physics Communications **182**(5), 1183 (2011). DOI https://doi.org/10.1016/j.cpc.2011.01.013. URL http://www.sciencedirect.com/science/article/pii/S0010465511000336
19. J.R. Hester, Journal of Applied Crystallography **39**(4), 621 (2006). DOI 10.1107/S0021889806015627. URL https://doi.org/10.1107/S0021889806015627
20. S. Gražulis, D. Chateigner, R.T. Downs, A.F.T. Yokochi, M. Quirós, L. Lutterotti, E. Manakova, J. Butkus, P. Moeck, A. Le Bail, Journal of Applied Crystallography **42**(4), 726 (2009). DOI 10.1107/S0021889809016690. URL http://dx.doi.org/10.1107/S0021889809016690
21. S. Graulis, A. Dakevi, A. Merkys, D. Chateigner, L. Lutterotti, M. Quirs, N.R. Serebryanaya, P. Moeck, R.T. Downs, A. Le Bail, Nucleic Acids Research **40**(D1), D420 (2012). DOI 10.1093/nar/gkr900. URL http://nar.oxfordjournals.org/content/40/D1/D420.abstract
22. H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne, Nucleic Acids Research **28**(1), 235 (2000). DOI doi:10.1093/nar/28.1.235. URL http://dx.doi.org/10.1093/nar/28.1.235
23. T. Hamelryck, B. Manderick, Bioinformatics **19**(17), 2308 (2003). DOI 10.1093/bioinformatics/btg299. URL http://dx.doi.org/10.1093/bioinformatics/btg299
24. A.H. Larsen, J.J. Mortensen, J. Blomqvist, I.E. Castelli, R. Christensen, M. Duak, J. Friis, M.N. Groves, B. Hammer, C. Hargus, E.D. Hermes, P.C. Jennings, P.B. Jensen, J. Kermode, J.R. Kitchin, E.L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J.B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schitz, O. Schtt, M. Strange, K.S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, K.W. Jacobsen, Journal of Physics: Condensed Matter **29**(27), 273002 (2017). URL http://stacks.iop.org/0953-8984/29/i=27/a=273002
25. R.W.G. Wyckoff, vol. 1 (Interscience Publishers, 1963)
26. A. Togo. spglib : finding and handling crystal symmetries (2009-2018). URL https://atztogo.github.io/spglib/
27. R.W. Grosse-Kunstleve, Acta Crystallographica Section A **55**(2 Part 2), 383 (1999). DOI 10.1107/S0108767398010186. URL https://doi.org/10.1107/S0108767398010186
28. E.J. Kirkland, *Advanced Computing in Electron Microscopy*, 2nd edn. (Springer, New York, 2010). DOI 10.1007/978-1-4419-6533-2. URL http://link.springer.com/10.1007/978-1-4419-6533-2
29. J.M. COWLEY, in *Diffraction Physics (Third Edition)*, ed. by J.M. COWLEY, third edition edn., North-Holland Personal Library (North-Holland, Amsterdam, 1995), pp. 231 – 254. DOI https://doi.org/10.1016/B978-044482218-5/50013-4. URL http://www.sciencedirect.com/science/article/pii/B9780444822185500134
30. E.J. Kirkland, R.F. Loane, J. Silcox, Ultramicroscopy **23**(1), 77 (1987). DOI https://doi.org/10.1016/0304-3991(87)90229-4. URL http://www.sciencedirect.com/science/article/pii/0304399187902294
31. C. Xiao, C. Jin, X. Wang, Materials Chemistry and Physics **111**(2), 209 (2008). DOI https://doi.org/10.1016/j.matchemphys.2008.01.020. URL http://www.sciencedirect.com/science/article/pii/S0254058408000102
32. T. Ida, M. Ando, H. Toraya, Journal of Applied Crystallography **33**(6), 1311. DOI 10.1107/S0021889800010219. URL https://onlinelibrary.wiley.com/doi/abs/10.1107/S0021889800010219
33. S.G. Tomlin, Journal of Physics D: Applied Physics **1**(12), 1667 (1968). URL http://stacks.iop.org/0022-3727/1/i=12/a=312
34. L. Greengard, J.Y. Lee, SIAM Review **46**(3), 443 (2004). DOI 10.1137/S003614450343200X. URL https://doi.org/10.1137/S003614450343200X
35. A. Pawlik, J. Segal, H. Sharp, M. Petre, Computing in Science Engineering **17**(1), 28 (2015). DOI 10.1109/MCSE.2014.93
36. A. Pryor, C. Ophus, J. Miao, Advanced Structural and Chemical Imaging **3**(1), 15 (2017). DOI 10.1186/s40679-017-0048-z. URL https://doi.org/10.1186/s40679-017-0048-z